PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-286896
(43)Date of publication of application : 01.11.1996

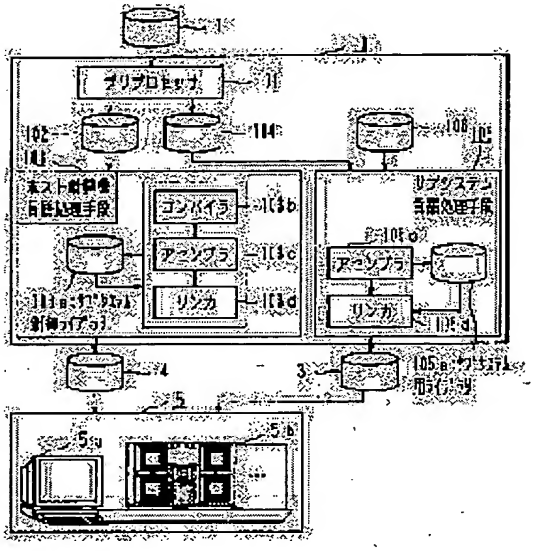
(51)Int.Cl. G06F 9/06
G06C 15/22
G06F 9/45
G06F 11/28
G06F 15/16

(21)Application number : 07-089255
(22)Date of filing : 14.04.1995

(71)Applicant : MITSUBISHI ELECTRIC CORP
(72)Inventor : TSUBOTA HIRONO
TAMURA TOSHIYUKI
TANAKA KENICHI
KOMORI NOBUFUMI

(54) SOFTWARE DEVELOPMENT METHOD AND SOFTWARE DEVELOPMENT SYSTEM

(57)Abstract:
PURPOSE: To obtain a software development method and a software development system in which a software program executed in a target system comprising plural processors is efficiently developed from a program described in a high class language.
CONSTITUTION: A pre-processor 101 is used to enter a source program 1, it is decoded and the processing in the program is divided into sub programs and a processor of execution is selected. Language processing means 103, 105 generate execution programs 3, 4 operated by each processor based on the divided sub programs. Furthermore, a debugger 6 has a simulator 6b simulating the operation of a target system 5 and two execution systems for the target system 5 and stores information on the way of processing to select an execution system by a command of a user interface 6a.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平8－286896

(43)公開日　平成８年(1996)11月１日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/06	５３０		Ｇ０６Ｆ	9/06	５３０Ａ
Ｇ０６Ｃ	15/22		7368－5E	Ｇ０６Ｃ	15/22	
Ｇ０６Ｆ	9/45		7313－5B	Ｇ０６Ｆ	11/28	Ａ
	11/28				15/16	４３０Ａ
	15/16	４３０	7737－5B		9/44	３２２Ｂ

審査請求　未請求　請求項の数18　ＯＬ　（全 20 頁）

(21)出願番号　特願平7－89255

(22)出願日　平成７年(1995)４月14日

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号

(72)発明者　坪田　浩乃
尼崎市塚口本町八丁目１番１号　三菱電機株式会社半導体基礎研究所内

(72)発明者　田村　俊之
尼崎市塚口本町八丁目１番１号　三菱電機株式会社半導体基礎研究所内

(72)発明者　田中　健一
尼崎市塚口本町八丁目１番１号　三菱電機株式会社半導体基礎研究所内

(74)代理人　弁理士　高田　守　（外４名）

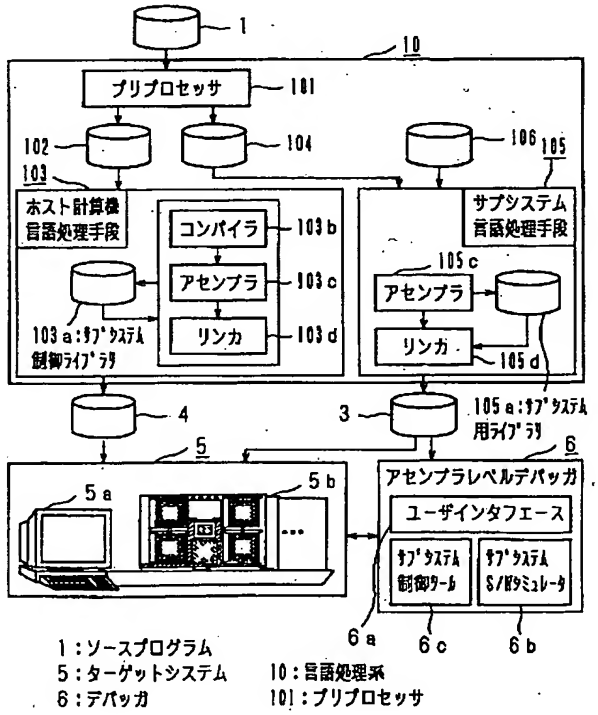
最終頁に続く

(54)【発明の名称】　ソフトウェア開発方法及びソフトウェア開発システム

(57)【要約】

【目的】　高級言語で記述されたプログラムから、複数のプロセッサで構成されるターゲットシステムで実行されるソフトウェアを効率よく開発できるソフトウェア開発方法及びソフトウェア開発システムを得る。

【構成】　プリプロセッサ１０１によって、ソースプログラム１を入力し、これを解釈してその中の処理をサブプログラムに分割し、実行すべきプロセッサを選択する。言語処理手段１０３，１０５によって、分割したサブプログラムに基き、各プロセッサで動作する実行プログラム３，４を生成する。また、デバッガ６は、ターゲットシステム５の動作を模擬するシミュレータ６ｂとターゲットシステム５の２つの実行系を有し、ユーザインタフェース６ａの指示により、処理途中の情報を保持して実行系を切り替える。



1

【特許請求の範囲】

【請求項１】　複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込んで複数のサブプログラムに分割し、この分割したサブプログラムのそれぞれを上記複数のプロセッサのうちのどのプロセッサで実行させるかを割り当てることを特徴とするソフトウェア開発方法。

【請求項２】　サブプログラムに分割する際、ソースプログラムを読み込んで文法解析し、データ依存関係を解析して種々の最適化を行い、最適化後のプログラムを、複数のプロセッサが分担する実行可能なサブプログラムに分割することを特徴とし、上記サブプログラムを割り当てる際、上記分割したサブプログラムのそれぞれを上記複数のプロセッサのうちのどのプロセッサで実行させるかを決定し、上記ソースプログラムの分割に伴って必要となるプログラムと上記サブプログラムに基づいて、上記プロセッサのそれぞれで実行すべき実行プログラムを生成することを特徴とする請求項１記載のソフトウェア開発方法。

【請求項３】　実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、上記プロセッサ毎のファイルに生成することを特徴とする請求項２記載のソフトウェア開発方法。

【請求項４】　実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、同一のファイルに生成することを特徴とする請求項２記載のソフトウェア開発方法。

【請求項５】　複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込み、このソースプログラムを解釈してその中の記述を主プログラムと少なくとも１つのサブプログラムに分割し、上記複数のプロセッサのなかで上記分割したプログラムのそれぞれを実行すべきプロセッサを選択するプリプロセッサ、及び上記分割したプログラムから上記プロセッサで動作する実行プログラムを生成する少なくとも１つの言語処理部を備えたことを特徴とするソフトウェア開発システム。

【請求項６】　ターゲットシステムの少なくとも１つのプロセッサで実行する処理に対応する手続きを予め格納したライブラリを備え、プリプロセッサは、複数のプロセッサのなかでサブプログラムを実行すべきプロセッサを選択した後、上記選択されたプロセッサで実行する処理に応じて、上記ライブラリから手続きを選択し、ソースプログラムの上記プロセッサで実行する処理の部分を、その処理を駆動する手続き呼出し文に置き換えると共に、上記選択された手続きの名前のリストを出力することを特徴とする請求項５記載のソフトウェア開発システム。

【請求項７】　言語処理部が、ターゲットシステムの少

2

なくとも１つのプロセッサの実行プログラムを生成する際、生成された上記実行プログラムに内在するシンボルのアドレス情報の少なくとも一部をアドレス情報ファイルに出力し、主プログラム部分の実行プログラムを生成する際に上記アドレス情報ファイルを参照して上記実行プログラムにアドレス情報を含める手段を有することを特徴とする請求項５または請求項６記載のソフトウェア開発システム。

【請求項８】　ターゲットシステムは、特定のプロセッサであるホストプロセッサとその他のプロセッサであるサブシステムで構成されるものとし、プリプロセッサは、入力されたソースプログラムを解釈して上記ホストプロセッサで実行すべき主プログラムと、上記サブシステムで実行すべきサブプログラムに分割し、言語処理部は、上記プリプロセッサから出力された手続きの名前のリストに基づいて上記サブシステム用の実行プログラムを生成するサブシステム用言語処理手段と、上記主プログラムに基づいて上記ホストプロセッサ用の実行プログラムを生成するホストプロセッサ用言語処理手段で構成されたことを特徴とする請求項６または請求項７記載のソフトウェア開発システム。

【請求項９】　複数のプロセッサの１つは、プロセス管理とファイル管理の少なくともいずれかの管理を主な処理とするホスト計算機であり、その他の上記プロセッサの少なくとも１つは、演算処理を主な処理とするサブシステムであることを特徴とする請求項５ないし請求項８のいずれかに記載のソフトウェア開発システム。

【請求項１０】　複数のプロセッサの１つは、汎用の計算機であり、その他の上記プロセッサの少なくとも１つは、上記計算機のシステムバスに接続するサブシステムであることを特徴とする請求項５ないし請求項８のいずれかに記載のソフトウェア開発システム。

【請求項１１】　複数のプロセッサの１つは、汎用の計算機であり、その他の上記プロセッサの少なくとも１つは、上記計算機のシステム拡張用の汎用バスに接続するサブシステムであることを特徴とする請求項５ないし請求項８のいずれかに記載のソフトウェア開発システム。

【請求項１２】　ターゲットシステムの動作を模擬するシミュレータと、上記シミュレータと上記ターゲットシステムの間で上記プログラムの実行系を切り替える切り替え手段と、上記プログラムの実行の開始時または実行の途中で実行系の切り替えを指示するユーザインターフェースとを有するデバッガを備えたことを特徴とするソフトウェア開発システム。

【請求項１３】　切り替え手段は、ターゲットシステムがプログラムを実行する過程で更新する可能性の有る記憶手段の内容の全部を保存し、上記プログラムの実行系を切り替える際に、上記保存した記憶手段の内容を切り替え先の記憶手段に格納する機能を有することを特徴とする請求項１２に記載のソフトウェア開発システム。

【請求項１４】　切り替え手段は、ターゲットプロセッサがプログラムを実行する過程で更新する可能性の有る記憶手段の内容のうち更新されたものを保存し、上記プログラムの実行系を切り替える際に、上記保存した記録手段の内容のうちで最新の内容を切り替え先の記憶手段に格納する機能を有することを特徴とする請求項１２に記載のソフトウェア開発システム。

【請求項１５】　プログラムの実行に伴って更新された記憶手段の内容を保存する機能と、この保存した内容を上記記憶手段に格納する機能を有するデバッガを備えたことを特徴とするソフトウェア開発システム。

【請求項１６】　複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、特定の指示を上記指定されたプロセッサに対してのみ実行する機能を有するデバッガを備えたことを特徴とするソフトウェア開発システム。

【請求項１７】　複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、上記指定されたプロセッサに関する情報のみを表示する機能を有するデバッガを備えたことを特徴とするソフトウェア開発システム。

【請求項１８】　デバッガ実行時の出力情報をファイルに格納するログファイル機能と、上記出力情報を画面に表示する表示機能を有し、上記ログファイル機能の動作時に、上記表示機能を抑止し得るデバッガを備えたことを特徴とするソフトウェア開発システム。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】この発明は、複数のプロセッサを有する高速・高性能計算機システムをターゲットとし、このターゲットシステム上で動作するソフトウェアを開発する方法及び開発システムに関するものである。

【０００２】

【従来の技術】従来、複数のプロセッサを有する計算機システムの一例として、ベクトル計算機システムがある。これは、通常のデータ処理を行ういわゆるスカラ演算を行うプロセッサと、これとは別にベクトル処理機構を行うプロセッサを有するものである。そして、ベクトル処理を得意とするプロセッサを用いて、行列演算のような規則的な演算を処理することにより、演算を高速に実行するものである。ベクトル計算機システムについては、文献（「Computer Architecture: A Quantitative Approach」、D.A.Patterson，J.L.Hennessy著、1990年、Morgan Kaufmann publishers 社刊）の第３５０頁～第３９７頁に詳細に開示されている。図８は、この文献に記載されているベクトル計算機システムにおけるベクトル処理を行うプロセッサの構成例を示すブロック図である。

【０００３】以下、図８に基づいてベクトル処理機構について説明する。図において、１００１はメインメモリ、１００２は複数のベクトルレジスタより構成されるベクトルレジスタ群、１００３はメインメモリ１００１とベクトルレジスタ群１００２の間でデータを転送をするベクトルロード／ストアユニット、１００４から１００８はベクトル演算器で、ベクトルレジスタ群１００２に格納されたベクトルデータに対してそれぞれ異なる演算を行うものである。例えば、１００４は加算／減算を行うベクトル演算器、１００５は乗算を行うベクトル演算器、１００６は除算を行うベクトル演算器、１００７は整数演算を行うベクトル演算器、１００８は論理演算を行うベクトル演算器である。また、１００９はスカラレジスタファイルで、ベクトルデータと定数を演算する場合に定数を格納するレジスタである。

【０００４】このような構成のベクトル演算機構上で動作する実行プログラムの動作と実行プログラムの生成方法について説明する。以下に、高級言語で記述されたはプログラムの一例を示す。

```
do 10 i=1,100
Y(i) = a*X(i) + Y(i)
10 continue
```

これはループ構造のプログラムであり、ループボディの処理は、配列Ｘの各要素に定数ａを乗じた結果に、配列Ｙの対応する要素を加算し、その結果を再び配列Ｙに書き戻す処理を記述しており、通称ＳＡＸＰＹまたはＤＡＸＰＹと呼ばれ、リンパック　ベンチマーク（Linpack benchmark）と呼ばれるベクトル演算の性能を測定するためのベンチマークプログラムの要素を構成するコードである。

【０００５】図９は上記の処理をベクトル処理機構で実行するためのプログラムコードの機械語記述例である。この図を基にベクトル処理機構の基本的な動作について説明する。ベクトル処理機構は基本的に図９で示した機械語プログラムに記述された各行を実行する。機械語プログラムの各行毎のベクトル処理機構の動作を（１）～（６）に説明する。

（１）１行目の命令は定数ａをスカラレジスタファイル１００９の０番（Ｆ０）にロードする。

（２）２行目の命令はメインメモリのアドレスＲｘを先頭アドレスにして格納された配列Ｘ（ベクトルＸ）をベクトルレジスタ群１００２の１番（Ｖ１）にロードする。

（３）３行目の命令はベクトルレジスタＶ１に格納されたベクトルＸとスカラレジスタＦ０に格納された定数ａの乗算をベクトルＸの各要素に対して実行し、ベクトルレジスタＶ２に格納する。

（４）４行目の命令はアドレスＲｙを先頭アドレスとして格納された配列Ｙ（ベクトルＹ）をメインメモリからベクトルレジスタＶ３にロードする。

（５）５行目の命令はベクトルレジスタＶ２に格納された途中結果のベクトルａ＊Ｘと４行目でロードしたベクトルＹの各々の要素同士の加算を行い、その結果をベクトルレジスタＶ４に格納する。

（６）６行目はベクトルレジスタＶ４に格納されたベクトルａ＊Ｘ＋Ｙを、再びメインメモリのアドレスＲｙ以降に連続的に書き戻す。

【０００６】なお、図９のなかで、各行の先頭カラムにはベクトル処理機構の命令を示したニーモニックが書かれており以下の意味を持つ。ＬＤはメインメモリ１００１の中の１データ語をスカラレジスタファイル１００９にロードする命令である。ＬＶはメインメモリ１００１に格納されたベクトルデータを転送する命令であり、一度に転送する要素数はベクトルレジスタ群１００２が格納できる要素数によって決定される。ＭＵＬＴＳＶは指定したスカラレジスタファイル１００９の内容と、指定したベクトルレジスタ群１００２に格納されたベクトルデータとの積を計算する命令である。ＡＤＤＶは２種類のベクトルデータを加算する命令を示している。ＳＶは指定したベクトルレジスタ群１００２に格納されたベクトルデータを先頭アドレスを指定してメインメモリ１００１に転送する命令を示している。また、第２カラムにはベクトル処理機構の命令を実行する際のオペランドが指定されている。

【０００７】さらに、Ｖ１～Ｖ４は、ベクトルレジスタ群１００２を構成し、各々ベクトルデータの要素を格納し、オペランドとして使用する場合は格納された各要素を連続的に読み出す／書き込むことができる。Ｆ０はスカラレジスタファイル１００９を構成する１スカラレジスタであり、Ｒｘ、Ｒｙはメインメモリのアドレスを指し示すアドレスポインタである。

【０００８】オペランドのうち少なくとも１つをベクトルレジスタにもつ命令（ベクトル演算命令）ＬＶ、ＭＵＬＴＳＶ、ＡＤＤＶ、ＳＶ等は１行の命令実行で通常、ベクトルレジスタに格納可能なデータの全てに対して演算を実行する。上記の例では（２）～（６）がベクトル演算の実行に対応し、各々の演算はベクトルＸとベクトルＹの要素数分だけ繰り返される。しかし、ベクトルレジスタ群１００２に格納できる要素数は有限であるため、ベクトルレジスタに格納可能な要素数を越えるベクトル演算を実行する場合は、上記の（１）～（６）の一連の処理を複数回繰り返すことにより実現される。これは通常ストリップ・マイニングと呼ばれる方法である。ベクトルレジスタ群１００２に格納可能なベクトルの要素数をＭ、実際に処理したいベクトルの要素数をＮ（Ｍ＜Ｎ）、（１）～（６）の一連の動作の繰返し回数をｎとすると、

ｎ　＝　Ｎ／Ｍ（ＮがＭで割り切れる場合）

または、

ｎ　＝　Ｎ／Ｍ＋１（ＮがＭで割り切れない場合）

という関係が成立する。

【０００９】また、このようなベクトル処理機構を有効に機能させるためには、機械語で記述することも可能であるが、プログラムの生産性、保守性等の観点から考えて現実的ではないのは明らかである。通常、高級言語で記述されたソースプログラムを解釈し、ベクトル処理機構を利用するための実行コードを生成する必要があり、一般にベクトル化と言われている。

【００１０】ベクトル化の方法の一例について簡単に説明する。ベクトル化は、ソースプログラム中から繰返しのループ構造を検出し、ループボディ内の変数を解釈し、ベクトルデータ、スカラデータ、インデックスデータに分類する。この変数の分類結果とデータ（ベクトルデータ、スカラデータ）の依存関係などをもとに、図９で示したようなベクトル命令列の生成が行われる。ここでいうデータの識別法は、ベクトル化を行う際に使用されるものであり、スカラデータとは、ループの繰返しを通じて変化しない変数を指し、インデックスデータとはループの繰返し毎に一定数だけ変化する変数を指し、ベクトルデータとは配列データのように複数のデータで構成され、ループの繰返し毎に配列のインデックスが一定の間隔で規則的に変化してアクセスされるデータ群のことを指す。

【００１１】

【発明が解決しようとする課題】上記のように、従来のこの種のベクトル計算機システムでは、配列データなどの構造化されたデータ群を処理の対象にしており、いわゆるベクトルデータを単位として処理を実行することにより演算器の利用効率の向上を図っている。構造化されたデータを定型的に処理するため、演算の実行タイミングはプログラムの実行前に確定している。このような状況で使用されることを前提として、ベクトル演算器１００４～１００８を多段にパイプライン化して高速処理を実現している。これは逆の見方をすると、ベクトル処理機構の高速処理を有効に使用することができるのは、配列等の定型のデータを対象に同一の処理を行うものに適用した場合に限られることになる。非定型的な処理を行う場合には、上記のようなベクトル処理機構を用いても効果が得られない。例えば、ループ構造を用いて記述され、配列データを規則的にアクセスするようなプログラムであってもループボディ内に条件分岐等の記述がなされている場合には、実行されるプログラムが実行中の条件によって変動するため、ベクトル化ができない。このように、ベクトル処理を行うことによって効果が得られるか否かは、記述されたソースプログラムの構造に依存しており、このため、ベクトル処理機構の高速処理の適用範囲はベクトル化が可能な構造をしたプログラムに限られるという問題点があった。また、近年、様々な得意とする分野を有するプロセッサが開発され、これらを効率よく利用するような汎用的なソフトウェア開発方法は

なかった。

【００１２】この発明は、上記のような課題を解決するためになされたものであり、高級言語で記述されたプログラムから、複数のプロセッサで構成されるターゲットシステムで実行されるソフトウェアを効率よく開発できる汎用的なソフトウェア開発方法及びソフトウェア開発システムを得ることを目的とするものである。

【００１３】

【課題を解決するための手段】この発明の請求項１に係るソフトウェア開発方法は、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込んで複数のサブプログラムに分割し、この分割したサブプログラムのそれぞれを複数のプロセッサのうちのどのプロセッサで実行させるかを割り当てることを特徴とするものである。

【００１４】また、この発明の請求項２に係るソフトウェア開発方法は、請求項１記載の発明に加え、サブプログラムに分割する際、ソースプログラムを読み込んで文法解析し、データ依存関係を解析して種々の最適化を行い；最適化後のプログラムを、複数のプロセッサが分担する実行可能なサブプログラムに分割することを特徴とし、サブプログラムを割り当てる際、分割したサブプログラムのそれぞれを複数のプロセッサのうちのどのプロセッサで実行させるかを決定し、ソースプログラムの分割に伴って必要となるプログラムとサブプログラムに基づいて、プロセッサのそれぞれで実行すべき実行プログラムを生成することを特徴とするものである。

【００１５】また、この発明の請求項３に係るソフトウェア開発方法は、請求項２記載の発明に加え、実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、プロセッサ毎のファイルに生成することを特徴とするものである。

【００１６】また、この発明の請求項４に係るソフトウェア開発方法は、請求項２記載の発明に加え、実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、同一のファイルに生成することを特徴とするものである。

【００１７】また、この発明の請求項５に係るソフトウェア開発システムは、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込み、このソースプログラムを解釈してその中の記述を主プログラムと少なくとも１つのサブプログラムに分割し、複数のプロセッサのなかで分割したプログラムのそれぞれを実行すべきプロセッサを選択するプリプロセッサ、及び分割したプログラムからプロセッサで動作する実行プログラムを生成する少なくとも１つの言語処理部を備えたものである。

【００１８】また、この発明の請求項６に係るソフトウェア開発システムは、請求項５記載の発明に加え、ターゲットシステムの少なくとも１つのプロセッサで実行する処理に対応する手続きを予め格納したライブラリを備え、そのプリプロセッサは、複数のプロセッサのなかでサブプログラムを実行すべきプロセッサを選択した後、選択されたプロセッサで実行する処理に応じて、ライブラリから手続きを選択し、ソースプログラムのプロセッサで実行する処理の部分を、その処理を駆動する手続き呼出し文に置き換えると共に、選択された手続きの名前のリストを出力することを特徴とするものである。

【００１９】また、この発明の請求項７に係るソフトウェア開発システムは、請求項５または請求項６記載の発明に加え、その言語処理部が、ターゲットシステムの少なくとも１つのプロセッサの実行プログラムを生成する際、生成された実行プログラムに内在するシンボルのアドレス情報の少なくとも一部をアドレス情報ファイルに出力し、主プログラム部分の実行プログラムを生成する際にアドレス情報ファイルを参照して実行プログラムにアドレス情報を含める手段を有することを特徴とするものである。

【００２０】また、この発明の請求項８に係るソフトウェア開発システムは、請求項６または請求項７記載の発明に加え、ターゲットシステムを、特定のプロセッサであるホストプロセッサとその他のプロセッサであるサブシステムで構成されるものとし、プリプロセッサを、入力されたソースプログラムを解釈してホストプロセッサで実行すべき主プログラムと、サブシステムで実行すべきサブプログラムに分割するものとし、言語処理部を、プリプロセッサから出力された手続きの名前のリストに基づいてサブシステム用の実行プログラムを生成するサブシステム用言語処理手段と、主プログラムに基づいてホストプロセッサ用の実行プログラムを生成するホストプロセッサ用言語処理手段で構成したことを特徴とするものである。

【００２１】また、この発明の請求項９に係るソフトウェア開発システムは、請求項５ないし請求項８のいずれかに記載の発明に加え、複数のプロセッサの１つを、プロセス管理とファイル管理の少なくともいずれかの管理を主な処理とするホスト計算機とし、その他のプロセッサの少なくとも１つを、演算処理を主な処理とするサブシステムとしたことを特徴とするものである。

【００２２】また、この発明の請求項１０に係るソフトウェア開発システムは、請求項５ないし請求項８のいずれかに記載の発明に加え、複数のプロセッサの１つを、汎用の計算機とし、その他のプロセッサの少なくとも１つを、計算機のシステムバスに接続するサブシステムとしたことを特徴とするものである。

【００２３】また、この発明の請求項１１に係るソフトウェア開発システムは、請求項５ないし請求項８のいずれかに記載の発明に加え、複数のプロセッサの１つを、

汎用の計算機とし、その他のプロセッサの少なくとも１つを、計算機のシステム拡張用の汎用バスに接続するサブシステムとしたことを特徴とするものである。

【００２４】また、この発明の請求項１２に係るソフトウェア開発システムは、ターゲットシステムの動作を模擬するシミュレータと、シミュレータとターゲットシステムの間でプログラムの実行系を切り替える切り替え手段と、プログラムの実行の開始時または実行の途中で実行系の切り替えを指示するユーザインターフェースとを有するデバッガを備えたことを特徴とするものである。

【００２５】また、この発明の請求項１３に係るソフトウェア開発システムは、請求項１２記載の発明に加え、切り替え手段を、ターゲットシステムがプログラムを実行する過程で更新する可能性の有る記憶手段の内容の全部を保存し、プログラムの実行系を切り替える際に、保存した記憶手段の内容を切り替え先の記憶手段に格納する機能を有するようにしたことを特徴とするものである。

【００２６】また、この発明の請求項１４に係るソフトウェア開発システムは、請求項１２記載の発明に加え、切り替え手段を、ターゲットプロセッサがプログラムを実行する過程で更新する可能性の有る記憶手段の内容のうち更新されたものを保存し、プログラムの実行系を切り替える際に、保存した記録手段の内容のうちで最新の内容を切り替え先の記憶手段に格納する機能を有するようにしたことを特徴とするものである。

【００２７】また、この発明の請求項１５に係るソフトウェア開発システムは、プログラムの実行に伴って更新された記憶手段の内容を保存する機能と、この保存した内容を記憶手段に格納する機能を有するデバッガを備えたことを特徴とするものである。

【００２８】また、この発明の請求項１６に係るソフトウェア開発システムは、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、特定の指示を指定されたプロセッサに対してのみ実行する機能を有するデバッガを備えたことを特徴とするものである。

【００２９】また、この発明の請求項１７に係るソフトウェア開発システムは、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、指定されたプロセッサに関する情報のみを表示する機能を有するデバッガを備えたことを特徴とするものである。

【００３０】また、この発明の請求項１８に係るソフトウェア開発システムは、デバッガ実行時の出力情報をファイルに格納するログファイル機能と、出力情報を画面に表示する表示機能を有し、ログファイル機能の動作時に、表示機能を抑止し得るデバッガを備えたことを特徴

とするものである。

【００３１】

【作用】請求項１記載の発明によるソフトウェア開発方法は、ソースプログラムの処理内容の中で、ターゲットシステムを構成する複数のプロセッサのそれぞれに適した処理をサブプログラムとして分割し、プロセッサのそれぞれに割り当てる。

【００３２】また、この発明の請求項２によるソフトウェア開発方法は、ソースプログラムを文法解析し、データ依存関係を解析して最適化する。この状態で、ソースプログラムの処理内容の中で、ターゲットシステムを構成する複数のプロセッサのそれぞれに適した処理をサブプログラムとして分割する。プロセッサのそれぞれに割り当てたサブプログラムと、ソースプログラムを分割したことにより必要となるプログラム、例えば通信部分のプログラムとから、プロセッサのそれぞれで実行すべき実行プログラムを生成する。

【００３３】また、この発明の請求項３によるソフトウェア開発方法は、プロセッサのそれぞれに割り当てた実行プログラムを別々のファイルに生成するものである。

【００３４】また、この発明の請求項４によるソフトウェア開発方法は、プロセッサのそれぞれに割り当てた実行プログラムを同一のファイルに生成するものである。

【００３５】また、この発明の請求項５によるソフトウェア開発システムは、プリプロセッサによって、ソースプログラムを解釈してその中の記述をサブプログラムに分割し、複数のプロセッサのなかでサブプログラムの処理に適したプロセッサに割り当てる。さらに、言語処理系によって、プロセッサで動作する実行プログラムのコードを生成する。

【００３６】また、この発明の請求項６によるソフトウェア開発システムは、予めプロセッサで実行する処理の手続きをライブラリに作成しておく。プリプロセッサによって、選択したプロセッサに適した手続きをライブラリの中から探し出す。そして、ソースプログラム中のそのプロセッサで実行させることが決定した部分については、そのプロセッサでその手続きを駆動するための手続き文に置き換える。これと共に、使用する手続きの名前のリストとして関数リストを出力する。即ち、プロセッサで実行されるプログラムとして、その関数リストを生成している。

【００３７】また、この発明の請求項７によるソフトウェア開発システムは、言語処理部によって、実行プログラムを生成する際、少なくとも１つのプロセッサ用の実行プログラムに内在するシンボルのアドレス情報を、マップ情報として出力する。プロセッサで実行されるプログラムとして、そのマップ情報を生成している。

【００３８】また、この発明の請求項８によるソフトウェア開発システムは、ホストプロセッサとサブシステムで構成されるターゲットシステムで、プリプロセッサに

よって、ソースプログラムの処理内容を解析してホストプロセッサで実行するプログラムとサブシステムで実行するプログラムに分割する。さらに、サブシステム用言語処理手段で、プリプロセッサから出力された手続きのリストに基づいてサブシステム用の実行プログラムを生成する。また、ホストプロセッサ用言語処理手段で、ホストプロセッサで実行すべきソースプログラムに基づいてホストプロセッサ用の実行プログラムを生成する。

【００３９】また、この発明の請求項９におけるソフトウェア開発システムのターゲットシステムは、ホスト計算機としてプロセス管理やファイル管理などの処理，サブシステムでは実行不可能な処理，サブシステムで実行すると効率の悪いような処理を主要な処理とし、サブシステムは、演算処理を主要な処理とする。ソースプログラムの処理内容のなかで、演算処理の部分をサブシステムで実行するように実行プログラムを生成する。

【００４０】また、この発明の請求項１０によるソフトウェア開発システムのターゲットシステムは、汎用のエンジニアリングワークステーションやパーソナルコンピュータなどの計算機とシステムバスで接続されるサブシステムであり、エンジニアリングワークステーションやパーソナルコンピュータなどの計算機で実行する主プログラムと、サブシステムで実行するサブプログラムを生成する。

【００４１】また、この発明の請求項１１によるソフトウェア開発システムのターゲットシステムは、汎用のエンジニアリングワークステーションやパーソナルコンピュータなどの計算機とシステム拡張用の汎用バスで接続されるサブシステムであり、エンジニアリングワークステーションやパーソナルコンピュータなどの計算機で実行する主プログラムと、サブシステムで実行するサブプログラムを生成する。

【００４２】また、この発明の請求項１２によるソフトウェア開発システムのデバッガは、シミュレータでターゲットシステムの動作を模擬する実行系と、現実のターゲットシステムで動作させる実行系を構成し、切り替え手段で２つの実行系を適宜切り替える。

【００４３】また、この発明の請求項１３によるソフトウェア開発システムのデバッガは、実行中の記憶手段の内容の全部を保存し、実行系を切り替える際に、その内容を切り替え先の記憶手段に格納して、デバッグを続行する。

【００４４】また、この発明の請求項１４によるソフトウェア開発システムのデバッガは、実行中の記憶手段の内容のうちで更新した部分を保存し、実行系を切り替える際に、その内容を切り替え先の記憶手段に格納して、デバッグを続行する。

【００４５】また、この発明の請求項１５によるソフトウェア開発システムのデバッガは、記憶手段における実行状況を保存し、必要に応じて、保存された内容を記憶

手段に格納してデバックを再開する。

【００４６】また、この発明の請求項１６によるソフトウェア開発システムのデバッガは、ターゲットシステムを構成する複数のプロセッサの中で、一部のプロセッサを注目プロセッサとし、特定のコマンドは注目プロセッサにのみ適用させる。

【００４７】また、この発明の請求項１７によるソフトウェア開発システムのデバッガは、ターゲットシステムを構成する複数のプロセッサの中で、一部のプロセッサを注目プロセッサとし、情報の表示コマンドに対して、注目プロセッサについてのみ表示し、他のプロセッサについての表示を抑止する。

【００４８】また、この発明の請求項１８によるソフトウェア開発システムのデバッガは、表示処理を１つの関数で管理し、ＣＲＴの画面への出力とログファイルへの出力を、例えばフラグによって切り替える。

【００４９】

【実施例】

実施例１．以下、この発明の一実施例を図を用いて説明する。図１は、この発明の実施例１によるソフトウェア開発方法に係る処理の流れを示す説明図である。実際にプログラムを実行するターゲットシステムは、複数の同種または異種のプロセッサで構成されたものである。図において、１は、ファイルに格納され、例えばＣ言語などの高級言語で記述されたソースプログラム、２は言語処理系であり、ソースプログラム１を読み込んでサブプログラムに分割する機能と、システムを構成する複数のプロセッサのうちのどのプロセッサで各サブプログラムを実行させるべきかを決定する機能を有する。３は言語処理系２で出力される実行プログラムであり、この実施例では、複数のプロセッサのそれぞれに対応し、プロセッサ毎のファイルを生成している。

【００５０】次に、言語処理系２の詳細な処理の流れについて説明する。まず、ステップＳ１では、高級言語で記述されたソースプログラムを読み込み、文法解析処理を行う。この文法解析処理とは、例えば、ソースプログラムの記述が言語文法にかなっているかのチェックを行うと同時に内部フォーマットに変換していく。次にステップＳ２で、内部フォーマットに変換されたソースプログラムに対して、データ依存関係の解析を行い、種々の最適化処理を行う。ステップＳ３は、最適化後のソースプログラムを分割し、複数のプロセッサで分担して実行可能な複数のサブプログラムを生成する。さらに、ステップＳ４では、分割して生成した複数のサブプログラムをどのプロセッサの実行プログラムとして割り当てるかを決定する。ステップＳ３，Ｓ４で具体的には、システムを構成するプロセッサの組み合わせに応じて、最適化後のプログラムから、各プロセッサで実行させると有利な部分を抽出する。例えば、ターゲットとするシステムにベクトル演算プロセッサがあれば、独立にベクトル演

13

算可能な部分の抽出する。また、画像処理プロセッサがあれば画像処理部分の抽出等を行う。さらに、互いにデータ依存関係の少ない独立部分の抽出を行い、その部分のプログラムをサブプログラムに分割する。この分割されたプログラムをステップＳ４の割り当て決定部において実際にどのプロセッサに割り当てるかを決定する。この後、ステップＳ５で、ソースプログラムの分割に伴うデータ通信や実行の同期コード、及び各々のプロセッサで実行できるように実行プログラムのコードを生成し、それぞれのファイル３に格納する。但し、プロセッサ間のデータ通信や実行の同期に必要なアドレスやキーは、各プロセッサに実行プログラムをロードする時に決定する。

【００５１】以上のように、この実施例では、高級言語で記述されたプログラムを同種もしくは異種のプロセッサから構成されるマルチプロセッサシステムで、各プロセッサの機能を有効に動作させる実行プログラムを自動的に効率よく生成することができる。また、ソースプログラムのデータのフロー解析を行うことで、複数のプロセッサからなるシステムでのボトルネックとなることの多いデータ通信量を最適化した効率のよいプログラムを生成することができる。また、実行プログラムをプロセッサ毎のファイルに生成しているので、各プロセッサ毎の実行コードレベルで最適化を行う等の処理がしやすいソフトウェア開発方法が得られる。

【００５２】実施例２．図２は、この発明の実施例２によるソフトウェア開発方法に係る処理の流れを示す説明図である。図において、図１と同一符号は同一、または相当部分を示しており、出力する実行プログラム３を格納するファイルを１つとしている。実施例１では、ターゲットシステムの各々のプロセッサ用のコードを独立したファイルに生成する場合の実施例について記したが、この実施例では、１つのファイルにすべてのプロセッサ用の実行コードを格納するように構成している。

【００５３】以上のように、この実施例では、実施例１と同様、高級言語で記述されたプログラムを同種もしくは異種のプロセッサから構成されるマルチプロセッサシステムで効率よく動作させる実行プログラムを生成することができる。特に、１つのファイルにすべてのプロセッサ用の実行コードを格納するので、実行プログラムの管理が容易となる。

【００５４】実施例３．以下、この発明の実施例３によるソフトウェア開発システムを図を用いて説明する。図３は、実施例３によるソフトウェア開発システムの全体構成を示す説明図である。図において、１は、ファイルに格納され、例えばＣ言語などの高級言語で記述されたソースプログラム、１０は言語処理系であり、ソースプログラム１を読み込んでサブプログラムに分割する機能と、システムを構成する複数のプロセッサのうちのどのプロセッサで各サブプログラムを実行させるべきかを決

14

定する機能を有する。また、３はサブシステム実行プログラムを格納するファイル、４はホスト実行プログラムを格納するファイル、５はターゲットシステムであり、複数のプロセッサ、例えばホスト計算機５ａと複数のサブシステム５ｂとを有する。この実施例は、ターゲットシステム５で実行されるプログラムを開発するときのものであり、ターゲットシステム５は、複数のプロセッサで構成され、そのうちのホスト計算機５ａを特定のプロセッサとして動作させ、他に複数の同種または異種のサブシステム５ｂがバスによって接続されている。この実施例におけるホスト計算機５ａは、サブシステム５ｂでは実行不可能な処理や、サブシステム５ｂで実行すると効率の悪いような演算処理や、プロセス管理とファイル管理のうちの少なくとも１つの管理を主な処理とする。また、サブシステム５ｂは演算処理を主な処理とする。

【００５５】次に、言語処理系１０の詳細な構成について説明する。１０１はプリプロセッサで、入力された高級言語で記述されたソースプログラムを解釈し、そのソースプログラム中の記述をホスト計算機５ａで実行すべき部分と、ホスト計算機以外のプロセッサ、即ちサブシステム５ｂで実行すべき部分に分割または選択する。１０２は、プリプロセッサ１０１から出力され、ホスト計算機言語処理系１０３に入力される主プログラムで、この実施例ではホストソースプログラムと称する。ホスト計算機言語処理系１０３は、サブシステム制御ライブラリ１０３ａ，コンパイラ１０３ｂ，アセンブラ１０３ｃ，リンカ１０３ｄで構成され、ホストソースプログラム１０２を解釈し、ホスト実行プログラム４を生成する。

【００５６】１０４は、プリプロセッサ１０１から出力され、サブシステム言語処理系１０５に入力されるサブシステム関数リストで、サブシステム５ｂで実行すべき部分であると選択された関数のリストである。サブシステム言語処理系１０５は、サブシステム用ライブラリ１０５ａ，アセンブラ１０５ｃ，リンカ１０５ｄで構成され、サブシステム５ｂで実行すべく選択された関数のリストを読み込み、サブシステム５ｂで実行されるサブシステム実行プログラム３を生成する。１０６は、サブシステム５ｂ用のアセンブリ言語で記述されたソースファイルである。

【００５７】次に動作について説明する。高級言語で記述されたソースプログラム１がプリプロセッサ１０１に入力される。この言語処理系１０には、サブシステム５ｂで実行する処理に対応する手続きを予め格納したライブラリ１０５ａを備えている。また、ライブラリ１０５ａに格納されている手続きの名前は手続き名として予めプリプロセッサ１０１に通知されている。プリプロセッサ１０１は、ホスト計算機５ａで処理可能なホストソースプログラム１０２と、サブシステム５ｂで実行すべき関数のサブシステム関数リスト１０４を生成する。その

際、サブシステム５ｂで実行する処理に応じて手続き名を選択し、ソースプログラム１のなかでサブシステム５ｂで実行する処理の部分に、その処理を駆動するための手続き呼出し文として記述する。この処理を駆動するための手続き呼出し文とは、サブシステム５ｂで所望の動作をさせるようにホスト計算機５ａから制御するためのプログラムを含むライブラリ関数として選択したものを、関数呼出の文として記述するものである。

【００５８】ホスト計算機言語処理系１０３は、ホスト計算機５ａで処理可能なホストソースプログラム１０２を読み込み、コンパイラ１０３ｂ，アセンブラ１０３ｃ，リンカ１０３ｄ，サブシステム制御ライブラリ１０３ａを用いて、ホスト計算機５ａ上で動作するホスト実行プログラム４を生成する。一方、サブシステム言語処理系１０５では、サブシステムアセンブラソースファイル１０６を読み込みアセンブラ１０５ｃでアセンブルする。また、サブシステム関数リスト１０４を読み込み、この中から実行に必要な関数群をサブシステム用ライブラリ１０５ａを参照してリンカ１０５ｄでリンクし、サブシステム実行プログラム３を生成する。アセンブラソースファイル１０６の入力が無い場合には、関数リスト１０４の関数群のみをリンカ１０５ｄでリンクして、サブシステム実行プログラム３を生成する。

【００５９】このようにソフトウェア開発システムを構成すれば、高級言語で記述されたソースプログラムを同種または異種のプロセッサから構成されるマルチプロセッサシステムで効率よく動作させるプログラムを生成することができる。また、プリプロセッサにより機能を実現するのでターゲットとするマルチプロセッサシステムのプロセッサごとに既に開発された処理系を有効に利用し、効率のよい実行プログラムを生成することができるので、短期間で所望の言語処理系を開発することができる。また、関数リストを生成するので、サブシステムの処理系として、ライブラリを用意するだけで所望の言語処理系が実現できる。特に、この実施例では、サブシステムがＯＳ等の有さないスレーブシステムである場合の言語処理系が実現できる。さらに、ターゲットシステムは、プロセス管理とファイル管理を主な処理とするホスト計算機と、演算処理を主な処理とするサブシステムで構成されており、サブシステムが簡単な構成でも高性能が実現できる。

【００６０】実施例４．以下、この発明の実施例４によるソフトウェア開発システムを図を用いて説明する。図４は、実施例４によるソフトウェア開発システムの構成を示す説明図である。図において、図３と同一符号は、同一、または相当部分を示している。さらに、１０７はサブシステム用オブジェクトマップ情報ファイルであり、サブシステム実行プログラム３に内在する手続き及び変数のアドレス情報を、ホストソースプログラム１０２に追加する際に参照する。３０１はサブシステム関数

のプロトタイピングを記述したインクルードファイルである。

【００６１】以下、この実施例による動作を図について説明する。Ｃ言語によるプログラムは一般に複数のソースプログラム１から構成される。各々のソースプログラム１はプリプロセッサ１０１に入力されると、例えばＣ言語によるホストソースプログラム１０２とサブシステムの関数リスト１０４を生成する。この実施例におけるサブシステム５ｂは、例えばＳＩＭＤタイプのマルチプロセッサシステムである。ソースプログラム１のループ実行部分を解析し、同時並列に演算実行できる部分を検出する。サブシステム５ｂで実行するのが有利であると判断した場合には、予め用意されたホスト計算機５ａ側の動作関数コールをホストソースプログラム中に生成する。さらに、サブシステム５ｂで動作させる関数をサブシステム関数リスト１０４に出力する。

【００６２】また、サブシステムで動作させる関数には、入力されたソースプログラム１中に関数コールの形で記述されているものもある。これらについては、サブシステム５ｂで実行可能な関数としてプロトタイピングされ、インクルードファイル３０１に宣言されている。プリプロセッサ１０１は、サブシステム用の関数宣言のインクルードファイル３０１を参照し、これに記述されている関数がソースプログラム１中にあるかどうかを判断し、ある場合はこれもサブシステム関数リスト１０４に出力する。

【００６３】サブシステム５ｂで関数動作させるためには、ホスト計算機５ａ側で必要なデータ転送を行い、その後、サブシステム５ｂを起動させる関数が動く。この起動時にサブシステム５ｂ上で動作させる関数の先頭アドレスをホスト計算機５ａからサブシステム５ｂに通知する。これに伴って、サブシステム５ｂで対応する関数が動作する。この実施例では、ホスト計算機５ａの関数名の各文字をすべて大文字にした関数名がサブシステム５ｂで動作する約束としている。これに対し、ホスト計算機５ａの関数名は、少なくとも１文字小文字を含まなくてはならないという約束とする。サブシステムリンカ１０５ｄは、サブシステム関数リスト１０４に記述された関数名に対して、大文字化された関数をサブシステムアセンブラソースファイル１０６またはサブシステムライブラリ１０５ａからさがしだしてリンクする。

【００６４】リンク後、サブシステム実行プログラム３の各関数の実行開始アドレスをサブシステム用オブジェクトマップ情報ファイル１０７に出力する。この情報ファイル１０７は、その一例を図５に示すようなＣ言語のヘッダファイルで、サブシステム側の各関数名をタグとしての先頭アドレスをｄｅｆｉｎｅ文で記述された形で、出力する。ホスト計算機Ｃコンパイラ１０３ｂの実行時には、このサブシステム用オブジェクトマップ情報ファイル１０７を参照し、各関数の先頭アドレスが定数

値としてホスト実行プログラム４に埋め込まれる。これにより、実行時に各関数の開始アドレスを知るために、テーブルをサーチする時間等が不要となり、高速実行が実現できる。一方、プリプロセッサ１０１のフロントエンドは、Ｃ言語のプリプロセッサおよび通常のＣ言語の文法解析処理部となっている。このため、サブシステム５ｂでの関数の先頭アドレスは、プリプロセッサ１０１に入力される時には、未定義となってしまう。また、まだ生成されていないヘッダファイルがインクルードされることも矛盾する。そこで、これらのアドレス用のラベルは、通常の関数と同様にｅｘｔｅｒｎ宣言をする約束としている。プリプロセッサ１０１の後段において、必要なインクルード文の挿入と、サブシステム５ｂでの手続き名である関数ラベルをシンボルテーブルから削除する作業を行う。このことで、プリプロセッサの入力も、Ｃ言語の文法解析処理部を利用することが可能となる。

【００６５】ターゲットシステム５のサブシステム５ｂは、汎用的なシステムである。このため、定型的な処理を実行するアクセラレータボードのように、予めサブシステム側のプログラムをＲＯＭ等で持たせておくことができない。即ち、ターゲットシステム５では、サブシステム５ｂで動作させるサブシステム実行プログラム３をホスト計算機５ａからロードする必要がある。この際、ホスト計算機５ａとサブシステム５ｂとのデータ転送の方法としては、ランダム転送やＤＭＡ転送等が使用できる。処理速度を考慮した場合、処理全体のなかでデータ転送がネックとなることが多いので、高速に転送ができるＤＭＡ転送を使用することが有利となる。このＤＭＡ転送では、初期オーバヘッドが大きいため、なるべく転送回数を減らし、例えば一度で転送することが好ましい。この実施例において、言語処理系１０でサブシステム実行プログラム３を生成した時、既に、サブシステムで動作させる関数があらかじめ決定しているので、実行の最初に一度だけサブシステム実行プログラム３をＤＭＡ転送でホスト計算機５ａからサブシステム５ｂにロードする。ホスト計算機５ａ上のプログラムで、サブシステム５ｂでの関数実行コールが発行され、その度に、ダイナミックにホスト計算機５ａからサブシステム５ｂにプログラムをロードするシステムに比べて、この実施例のものは一度だけのＤＭＡ転送で済むために、転送時間が少なくてすむ。また、２回以上実行される関数の再ロードも不要となるので、全体的な実行時間を減少させることができる。

【００６６】また、ユーザがサブシステム用のアセンブラ１０５ｃを使用して新たに開発した関数については、ユーザは、ホスト計算機５ａ側の動作についても記述を行う必要がある。このホスト計算機５ａ側に記述を行う動作とは、例えばデータ転送、サブシステム５ｂの起動、サブシステム５ｂの動作終了通知の認識及びサブシステム５ｂからホスト計算機５ａへの結果データの転送

等である。この際、サブシステム起動関数では、起動関数の関数名が第１引数として指定されることになっているので、プリプロセッサ１０１はサブシステム起動関数が入力したソースプログラム１中にあると判断した場合には、この関数名も関数リスト１０２に出力する。これについては、最初からすべて大文字の関数名とする。また、Ｃ言語ソースファイル中ではｅｘｔｅｒｎ宣言されるので、これについてもプリプロセッサ１０１の後段でシンボルテーブルから削除する。

【００６７】このように、この実施例では、サブシステム用オブジェクトマップ情報ファイル１０７を作成するので、実施例３の効果に加えて、実行時には、既に、サブシステムで動作させる関数があらかじめ決定しているので、サブシステムの実行プログラムを初期ロードできる。これにより１回のＤＭＡ転送でプログラムのロードが完了するため、転送時間が少なくてすみ、全体として、実行時間が少なくてすむ。

【００６８】なお、上記サブシステム用オブジェクトマップ情報ファイル１０７として出力する情報は、マップ情報として関数の先頭アドレスのみを含むものでもよく、また、処理する変数のアドレス情報もマップ情報に含むものでもよく、上記実施例と同様の効果を奏することができる。

【００６９】実施例５．図６は、この発明の実施例５に係るソフトウェア開発システムの構成を示す説明図である。図において、６はデバッガで、例えばアセンブラレベルデバッガであり、６ａはユーザインタフェース、６ｂはシミュレータで、例えばＳ／Ｗシミュレータ、６ｃはサブシステム制御ツールである。図中、図３と同一符号は同一または相当部分を示している。この実施例では、実施例３に加え、共通のユーザインターフェース６ａを有し、プログラム実行系を実際のターゲットシステム５とＳ／Ｗシミュレータ間で切り替える手段を有するアセンブラレベルデバッガ６を備えた構成になっている。

【００７０】このアセンブラレベルデバッガ６について詳細に説明する。アセンブラレベルデバッガ６は、ソフトウェア開発の際、ソースファイルを参照しながら会話形式でデバッグができる環境を提供するものである。ｕｎｉｘ等の一般的なデバッガの機能を有し、デバッグ対象プログラムの指定、ステップ実行、ブレイクポイントの設定、各レジスタ、メモリの値の表示と変更などのコマンドをサポートする。さらに、この実施例によるアセンブラレベルデバッガ６は、２つの実行系を構成している。即ち、実際にプログラムを実行するターゲットシステム５のサブシステム５ｂで実行させる場合と、サブシステム５ｂの動作を模擬するサブシステムＳ／Ｗシミュレータ６ｂで実行させる場合がある。そして、ユーザインターフェース６ａによって、ユーザは２つの実行系のいずれで実行させるかを、実行開始時または実行途中に

選択し会話形式で指示できる。以下、ターゲットシステム５のサブシステム５ｂで実行させる場合の実行系をＨ／Ｗによる実行系と称する。

【００７１】アセンブラレベルデバッガ６の切り替え手段は、ユーザがユーザインターフェース６ａで実行系を例えばサブシステム５ｂからサブシステムＳ／Ｗシミュレータ６ｂに変更指示した場合、サブシステム制御ツール６ｃを介してサブシステム５ｂで実行していた記憶手段の情報を保持し、サブシステムＳ／Ｗシミュレータ６ｂに切り替えて実行させる。

【００７２】以下、実行系を切り替える際の更に詳しい動作，機能について説明する。デバッガでは、特に実行系がＳ／Ｗシミュレータの場合の実行時間が問題となる。数時間あるいは数日かかる大規模のプログラムを実行する場合に、プログラムのある時点まで、あるいは数ステップ前に戻って実行をやり直したい場合に、プログラムの最初から再実行するのは、たいへん時間がかかることになり、大きなロスとなる。このため、この実施例によるデバッガ６では、数十ステップ前の状態にもどるアンドゥ機能、及び各時点での、シミュレータのレジスタを含む記憶手段のすべての状態のセーブ，リストア機能を有し、効率的なデバッグを実現している。

【００７３】セーブ機能は，プログラムの実行に伴って更新される可能性のある記憶手段の内容の少なくとも一部を保存するものである。記憶手段の全部を保存すると、データ保存用に必要な領域は極めて大きくなる。また、メモリのすべてを利用していない場合も多い。そこで、この実施例では、今までアクセスした最大アドレスを記憶しておき先頭アドレスから最大アクセスアドレスまでを保存するようにしている。また、サブシステムでの実行時には、保存すべきデータがさらに大きくなるこのため、０以外のデータのところだけ記憶させる。この他に、圧縮したデータを覚えさせるモードや、複数回保存する場合には、前回の保存するした情報に対して変更した部分だけ保存するモードなども備えて、効率よく必要な情報を保存するように構成している。リストア機能は、セーブ機能で保存した内容を再び記憶手段に格納するものである。

【００７４】このセーブ機能、リストア機能を介して、実行系を切り替える。デバッグ作業を続けていくと詳細にデバッグしたい範囲は、次第にせばまってくる。その際に、実行の速度は遅いが、実行系をＳ／Ｗシミュレータ６ｂにすることで、以下の２つの利点がある。

（１）パイプラインに隠れた通常は読み書きできないレジスタを読み書きすることが可能となる。

（２）実行系を実際のサブシステムにすると、デバッグする際にはデバッグ動作をさせるために、不要な割り込み処理等が含まれる。このため、実際の動作時と異なる動作をすることになり、再現できない動作を忠実に再現するＳ／Ｗシミュレータで詳細トレース等をとりながら

デバッグすることが可能となる。詳細にデバッグしたい範囲が終了した場合には、また、Ｈ／Ｗで高速にデバッグ作業をすすめるようにすれば、全体としてデバッグ時間を短縮することができる。但し、Ｈ／Ｗは高速化のためパイプライン化されていることがあり、このための処理が必要となる。この実施例では３段のパイプラインのＨ／Ｗを想定している。即ち、Ｈ／Ｗは３段のパイプライン動作をしているために、Ｈ／Ｗ実行とＳ／Ｗシミュレータでの実行での切り替えを行う場合には、２サイクル分のいずれの場合にも、通常動作以外の動作が必要となる。

【００７５】このようにこの実施例によるデバッガを用いてソフトウェアを開発すれば、効率の良いデバッグが可能となる。特に実行系としてシミュレータを用いる場合と実機を使用する場合は、ユーザインタフェース６ａによるコマンドで切り替え可能となっているので、同一の操作でシミュレータと実機を操作でき、処理操作の習熟が容易となる。

【００７６】特に、切り替え手段は、ターゲットシステムがプログラムを実行する過程で更新する可能性の有る記憶手段の内容の全部を保存し、プログラムの実行系を切り替える際に、保存した記憶手段の内容を切り替え先の記憶手段に格納する機能を有するもののみとすれば、簡単に実現することができ、この場合でも全体のデバック時間は短縮される。

【００７７】また、切り替え手段は、ターゲットプロセッサがプログラムを実行する過程で更新する可能性の有る記憶手段の内容のうち更新されたもののみを保存し、プログラムの実行系を切り替える際に、保存した記録手段の内容のうちで更新した内容のみを切り替え先の記憶手段に格納するように構成すれば、実行系の切り替え時間を短縮でき、効率の良いデバッグが可能となる。さらに記憶させるデータ量がおさえられる。

【００７８】また、この実施例によるデバッガでは、切り替え手段で２つの実行系を切り替える機能を有しているが、これに限るものではなく、プログラムの実行に伴って更新された内容を保存する機能と、この保存した内容を記憶手段に格納する機能を有するデバッガを構成することもできる。このような構成のデバッガでも、効率の良いデバッグが可能となる。特に、長時間におよぶ実行時には、処理の途中段階で実行状況を保存できるので、何らかの理由でシステムが停止した場合や数ステップ前から設定を変更して再実行したい場合でも最初から処理をやり直す必要がなく、保存した時点から実行を再開することが可能となり全体としてデバッグの時間を短縮することができる。

【００７９】実施例６．この実施例では、実施例５に加え、デバッガ６に着目ボード機能とチップ指定機能を備えたものである。並列処理システムのソフトウェアデバッグでは、情報量が多すぎてデバッグの妨げになること

がある。そこで、特定のボードまたはチップにのみ着目してデバッグできるように、着目ボードやチップを指定する機能を有する。ユーザインタフェース６ａによってデバックの対象とするサブシステムのボートまたはチップを、デバッグ実行前またはデバック実行中にユーザが指定する。これによって、ユーザインタフェース６ａ内にある着目チップテーブル内に登録される。デバッガ６は着目チップテーブルに登録されたボードやチップでの実行部分を対象として、詳細トレース等をとる。また、コマンドに伴ってプロセッサ指定がなされなかった場合には、予め着目チップテーブルに登録されたプロセッサに対する情報のみを表示または設定する。

【００８０】この実施例では、着目したいプロセッサを予め指定し、指定されたプロセッサに対してのみ実行することにより、各コマンドで詳細にプロセッサを指定する必要がなく、様々な設定や表示ができ、簡単な操作で、効率良くデバッグできる。また、指定されたプロセッサに関する情報のみを表示することにより、多くのプロセッサに対する膨大なデータの表示が抑止されるので、情報の認識性が良く、効率良くデバッグできる。従って、この実施例によれば、デバッグ対象を細かく指定することにより、ソフトウェア開発の際、効率良くデバッグできるソフトウェア開発システムが得られる。

【００８１】実施例７．この実施例では、実施例５に加え、デバッガ６にログファイル機能を備えたものである。ログファイル機能は、入力コマンド及び結果出力を指定ファイルに記憶させるものである。結果出力先の指定には、例えば下記の３つのモードが用意されている。

モード１：ＣＲＴのみへの表示

モード２：ログファイルのみへの出力

モード３：ＣＲＴとログファイル両方への出力

通常のログファイル機能は、ＣＲＴに表示したものをログファイルに記録する機能である。これに対しこの実施例では、ターゲットシステムが高速・高性能計算機システムでありログも膨大なものになると想定される。このため、例えば、表示処理部を特定の表示関数で必ず実行するように構成する。そして、ＣＲＴへの表示が必要でないものには、会話形式で指定することによって、表示関数を例えばフラグなどを用いて呼び出すことで、ＣＲＴへの表示はやめてログファイルにのみ実行状況を格納する。この場合でも、入力コマンドのエコーバッグは行われる。また、これらのユーザインタフェース６ａはファイルからのコマンド入力、即ち、コマンドをマクロファイルに記述することができ、さらにマクロファイルのネスティングも可能である。

【００８２】この実施例では、デバッグ出力先を実行途中で自由に指定できる。これを利用して、例えば、デバッグ実行時にはＣＲＴの表示によって大きな流れを把握し、デバッグ実行後にログファイルに基づいて特定部分について詳しく解析することができ、効率よくデバッグ

できるソフトウェア開発システムが得られる。例えば、画面上にはコマンド入力だけを表示して、画面からデバッグの大きな流れが読み取り、詳細データの解析はファイルの形で行うなど、効率良くデバッグできる。

【００８３】実施例５～実施例７で説明したデバッガは、アセンブラレベルのデバッガであったが、これに限るものではなく、高級言語レベルのデバッガにも適用できる。

【００８４】実施例８．図７はこの発明の実施例８によるソフトウェア開発システムに係るターゲットシステムを示す構成図である。図７（ａ）は実施例３，４で示したものであり、ホスト計算機５ａとサブシステム５ｂとはシステムバス５ｃで接続されている。この場合、汎用のエンジニアリングワークステーション５ａと、エンジニアリングワークステーションのシステムバス５ｃに接続するサブシステム５ｂとで構成されている。この実施例は、ターゲットシステム５の他の例を示すもので、汎用のエンジニアリングワークステーション５ａと、エンジニアリングワークステーションのシステム拡張用の汎用バス５ｄに接続するサブシステムとで構成されている。

【００８５】システムバス５ｃに接続するものは、低コストでシステムが実現でき、システムバス５ｃを使用するので高速に動作させることができる。また、汎用バス５ｄを使用するシステムでは、サブシステム５ｂを様々なワークステーションやパーソナルコンピュータに接続することができる。なお、この実施例ではホスト計算機５ａとして、エンジニアリングワークステーションを用いた例を示したが、パーソナルコンピュータなどの汎用計算機全般を用いても同様の効果がある。また、この実施例ではターゲットシステムとして、ホスト計算機５ａとサブシステム通信手段としてバスで接続する場合について述べたが、例えばトーラスのようなマルチプロセッサを接続する接続網全般を用いても同様の効果が得られるのは明かである。

【００８６】また、実施例１～実施例８によるソフトウェア開発システムは、主にユーザＩ／Ｆ、ファイル管理などを司るホスト計算機とＳＩＭＤ型マルチプロセッサで構成されるサブシステムをターゲットシステムとした例のみを示したが、これに限るものではなく、ＭＩＭＤ型や、全く異なる型の複数のプロセッサからなる情報処理システム全般に適用でき、同様の効果が得られることは明らかである。

【００８７】

【発明の効果】以上のように、請求項１記載の発明によれば、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込んで複数のサブプログラムに分割し、この分割したサブプログラムのそれぞれを複数のプロセッサのうちのどのプロセッサで実行させるかを割り

当てることにより、高級言語で記述されたプログラムを同種または異種のプロセッサから構成されるマルチプロセッサシステムで、各プロセッサの機能を有効に動作させる実行プログラムを自動的に生成することができるソフトウェア開発方法が得られる効果がある。

【００８８】また、請求項２記載の発明によれば、請求項１記載の発明に加え、サブプログラムに分割する際、ソースプログラムを読み込んで文法解析し、データ依存関係を解析して種々の最適化を行い、最適化後のプログラムを、複数のプロセッサが分担する実行可能なサブプログラムに分割することを特徴とし、サブプログラムを割り当てる際、分割したサブプログラムのそれぞれを複数のプロセッサのうちのどのプロセッサで実行させるかを決定し、ソースプログラムの分割に伴って必要となるプログラムとサブプログラムに基づいて、プロセッサのそれぞれで実行すべき実行プログラムを生成することにより、請求項１の効果に加え、ソースプログラムのデータのフロー解析を行うことで、複数のプロセッサからなるシステムでのボトルネックとなることの多いデータ通信量を最適化した効率のよいプログラムを生成することができるソフトウェア開発方法が得られる効果がある。

【００８９】また、請求項３記載の発明によれば、請求項２記載の発明に加え、実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、プロセッサ毎のファイルに生成することにより、請求項２の効果に加え、各プロセッサ毎の実行コードレベルで最適化を行う等の処理がしやすいソフトウェア開発方法が得られる効果がある。

【００９０】また、請求項４記載の発明によれば、請求項２記載の発明に加え、実行プログラムを生成する際、複数のプロセッサのそれぞれで実行する実行プログラムを、同一のファイルに生成することにより、請求項２の効果に加え、実行プログラムの管理が容易となるソフトウェア開発方法が得られる効果がある。

【００９１】また、請求項５記載の発明によれば、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、ソースプログラムを読み込み、このソースプログラムを解釈してその中の記述を主プログラムと少なくとも１つのサブプログラムに分割し、複数のプロセッサのなかで分割したプログラムのそれぞれを実行すべきプロセッサを選択するプリプロセッサ、及び分割したプログラムからプロセッサで動作する実行プログラムを生成する少なくとも１つの言語処理部を備えたことにより、高級言語で記述されたプログラムを複数のプロセッサから構成されるマルチプロセッサシステムで効率よく動作させるプログラムを生成することができ、また、プリプロセッサにより機能を実現するのでターゲットシステムを構成するプロセッサごとに既に開発された処理系を有効に利用し効率のよい実行プログラムを生成することができ、短期間で所望の言語処理系を開発することができるソフトウェア開発システムが得られる効果がある。

【００９２】また、請求項６記載の発明によれば、請求項５記載の発明に加え、ターゲットシステムの少なくとも１つのプロセッサで実行する処理に対応する手続きを予め格納したライブラリを備え、プリプロセッサは、複数のプロセッサのなかでサブプログラムを実行すべきプロセッサを選択した後、選択されたプロセッサで実行する処理に応じて、ライブラリから手続きを選択し、ソースプログラムのプロセッサで実行する処理の部分を、その処理を駆動する手続き呼出し文に置き換えると共に、選択された手続きの名前のリストを出力することにより、請求項５の効果に加え、ターゲットとする複数のプロセッサごとに既に開発された処理系を有効に利用し効率のよい実行プログラムを生成することができ、また、関数のリストを生成するので、サブシステムの処理系としてライブラリを用意するだけで所望の言語処理系が実現できるソフトウェア開発システムが得られる効果がある。

【００９３】また、請求項７記載の発明によれば、請求項５または請求項６記載の発明に加え、言語処理部が、ターゲットシステムの少なくとも１つのプロセッサの実行プログラムを生成する際、生成された実行プログラムに内在するシンボルのアドレス情報の少なくとも一部をアドレス情報ファイルに出力し、主プログラム部分の実行プログラムを生成する際にアドレス情報ファイルを参照して実行プログラムにアドレス情報を含める手段を有することにより、請求項５または請求項６の効果に加え、実行時に１回の転送でサブシステムの実行プログラムをロードでき、転送時間が少なくてすみ、全体として、実行時間が少なくてすむソフトウェア開発システムが得られる効果がある。

【００９４】また、請求項８記載の発明によれば、請求項６または請求項７記載の発明に加え、ターゲットシステムは、特定のプロセッサであるホストプロセッサとその他のプロセッサであるサブシステムで構成されるものとし、プリプロセッサは、入力されたソースプログラムを解釈してホストプロセッサで実行すべき主プログラムと、サブシステムで実行すべきサブプログラムに分割し、言語処理部は、プリプロセッサから出力された手続き名のリストに基づいてサブシステム用の実行プログラムを生成するサブシステム用言語処理手段と、主プログラムに基づいてホストプロセッサ用の実行プログラムを生成するホストプロセッサ用言語処理手段で構成されたことにより、請求項６または請求項７の効果に加え、サブシステムが、ＯＳなどを有さないスレーブシステムである場合の言語処理系が実現できるソフトウェア開発システムが得られる効果がある。

【００９５】また、請求項９記載の発明によれば、請求項５ないし請求項８のいずれかに記載の発明に加え、複

数のプロセッサの１つは、プロセス管理とファイル管理の少なくともいずれかの管理を主な処理とするホスト計算機であり、その他のプロセッサの少なくとも１つは、演算処理を主な処理とするサブシステムであることにより、請求項５ないし請求項８のいずれかの効果に加え、サブシステムが簡単な構成でも高性能が実現できるソフトウェア開発システムが得られる効果がある。

【００９６】また、請求項１０記載の発明によれば、請求項５ないし請求項８のいずれかに記載の発明に加え、複数のプロセッサの１つは、汎用の計算機であり、その他のプロセッサの少なくとも１つは、計算機のシステムバスに接続するサブシステムであることにより、請求項５ないし請求項８のいずれかの効果に加え、低コストでシステムが実現でき、システムバスを使用するので高速に動作させることができるソフトウェア開発システムが得られる効果がある。

【００９７】また、請求項１１記載の発明によれば、請求項５ないし請求項８のいずれかに記載の発明に加え、複数のプロセッサの１つは、汎用の計算機であり、その他のプロセッサの少なくとも１つは、計算機のシステム拡張用の汎用バスに接続するサブシステムであることにより、請求項５ないし請求項８記載のいずれかの効果に加え、低コストでシステムが実現でき、汎用バスを使用するのでサブシステムを様々なワークステーションやパーソナルコンピュータに接続することができるソフトウェア開発システムが得られる効果がある。

【００９８】また、請求項１２記載の発明によれば、ターゲットシステムの動作を模擬するシミュレータと、シミュレータとターゲットシステムの間でプログラムの実行系を切り替える切り替え手段と、プログラムの実行の開始時または実行の途中で実行系の切り替えを指示するユーザインターフェースとを有するデバッガを備えたことにより、実行系としてシミュレータを用いる場合とＨ／Ｗを使用する場合を切り替えることができ、同一の操作でシミュレータとＨ／Ｗを操作することが可能となり処理操作の習熟が容易で、効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【００９９】また、請求項１３記載の発明によれば、請求項１２記載の発明に加え、切り替え手段は、ターゲットシステムがプログラムを実行する過程で更新する可能性の有る記憶手段の内容の全部を保存し、プログラムの実行系を切り替える際に、保存した記憶手段の内容を切り替え先の記憶手段に格納する機能を有することにより、請求項１２の効果に加え、実行系としてシミュレータを用いる場合とＨ／Ｗを使用する場合とを切り替えてデバッグでき、同一の操作でシミュレータとＨ／Ｗを操作することができるので処理操作の習熟が容易であり、２つの実行系を用いて自在にデバッグ作業をすることができ、時間的にも効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【０１００】また、請求項１４記載の発明によれば、請求項１２記載の発明に加え、切り替え手段は、ターゲットプロセッサがプログラムを実行する過程で更新する可能性の有る記憶手段の内容のうち更新されたものを保存し、プログラムの実行系を切り替える際に、保存した記録手段の内容のうちで最新の内容を切り替え先の記憶手段に格納する機能を有することにより、請求項１２の効果に加え、効率良くデバッグできると共に、記憶させるデータ量がおさえられるソフトウェア開発システムが得られる効果がある。

【０１０１】また、請求項１５記載の発明によれば、プログラムの実行に伴って更新された記憶手段の内容を保存する機能と、この保存した内容を記憶手段に格納する機能を有するデバッガを備えたことにより、処理の途中段階で実行状況を保存しておき、この保存した時点から実行を再開することが可能となり全体としてデバッグの時間を短縮することができ、効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【０１０２】また、請求項１６記載の発明によれば、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、特定の指示を指定されたプロセッサに対してのみ実行する機能を有するデバッガを備えたことにより、注目したいプロセッサを予め指定しておくことができるので、各コマンドで詳細にプロセッサを指定する必要がなく、様々な設定や表示ができ、簡単な操作で、効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【０１０３】また、請求項１７記載の発明によれば、複数のプロセッサを有するターゲットシステムで実行されるソフトウェアを開発するものにおいて、少なくとも１つのプロセッサを指定する手段と、指定されたプロセッサに関する情報のみを表示する機能を有するデバッガを備えたことにより、注目したいプロセッサを予め指定しておくことができるので、多くのプロセッサに対する膨大なデータの表示が抑止されるので、情報の認識性が良く、効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【０１０４】また、請求項１８記載の発明によれば、デバッガ実行時の出力情報をファイルに格納するログファイル機能と、出力情報を画面に表示する表示機能を有し、ログファイル機能の動作時に、表示機能を抑止し得るデバッガを備えたことにより、画面上にはコマンド入力だけが現れるので、画面からはデバッグの大きな流れが読み取れ、詳細データの解析はファイルの形で行えるので、効率良くデバッグできるソフトウェア開発システムが得られる効果がある。

【図面の簡単な説明】

【図１】　この発明の実施例１によるソフトウェア開発方法に係る言語処理系の処理の流れを示す説明図であ

る。

【図２】　この発明の実施例２によるソフトウェア開発方法に係る言語処理系の処理の流れを示す説明図である。

【図３】　この発明の実施例３によるソフトウェア開発システムの構成を示す説明図である。

【図４】　この発明の実施例４によるソフトウェア開発システムの構成を示す説明図である。

【図５】　実施例４に係るサブシステム用オブジェクトマップ情報ファイルの一例を示す説明図である。

【図６】　この発明の実施例５によるソフトウェア開発システムの構成を示す説明図である。

【図７】　この発明の実施例８によるソフトウェア開発システムに係るターゲットシステムを示す構成図である。

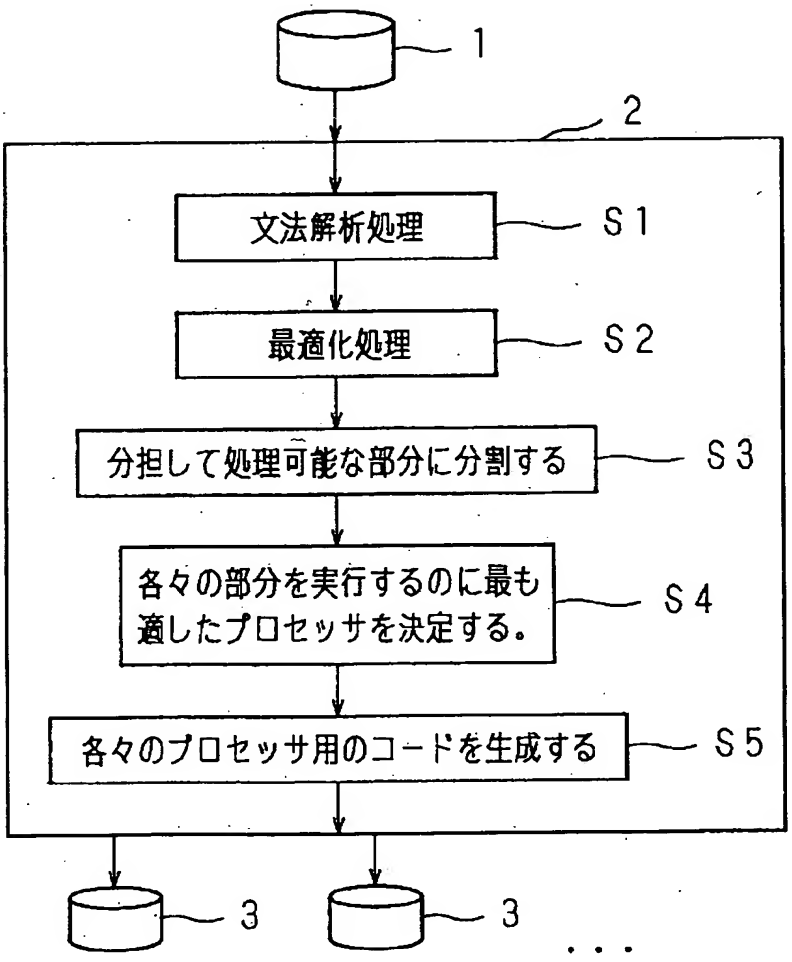
【図８】　従来のベクトル計算機システムにおけるベクトル処理を行うプロセッサの構成例を示すブロック図である。

【図９】　従来のベクトル処理機構で実行するためのプログラムコードの機械語記述例を示す説明図である。

【符号の説明】

１　ソースプログラム、３　サブシステム実行プログラム、４　ホスト実行プログラム、５　ターゲットシステム、６　デバッガ、１０　言語処理系、１０１プリプロセッサ、１０２　ホストソースプログラム、１０３　ホスト計算機言語処理手段、１０４　サブシステム関数リスト、１０５　サブシステム言語処理手段。

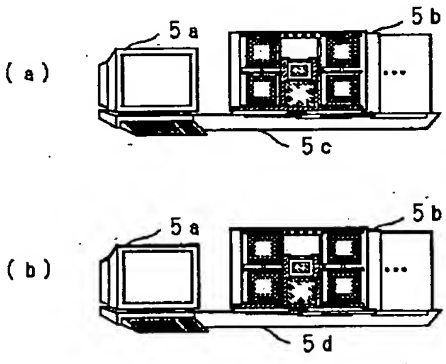
【図１】



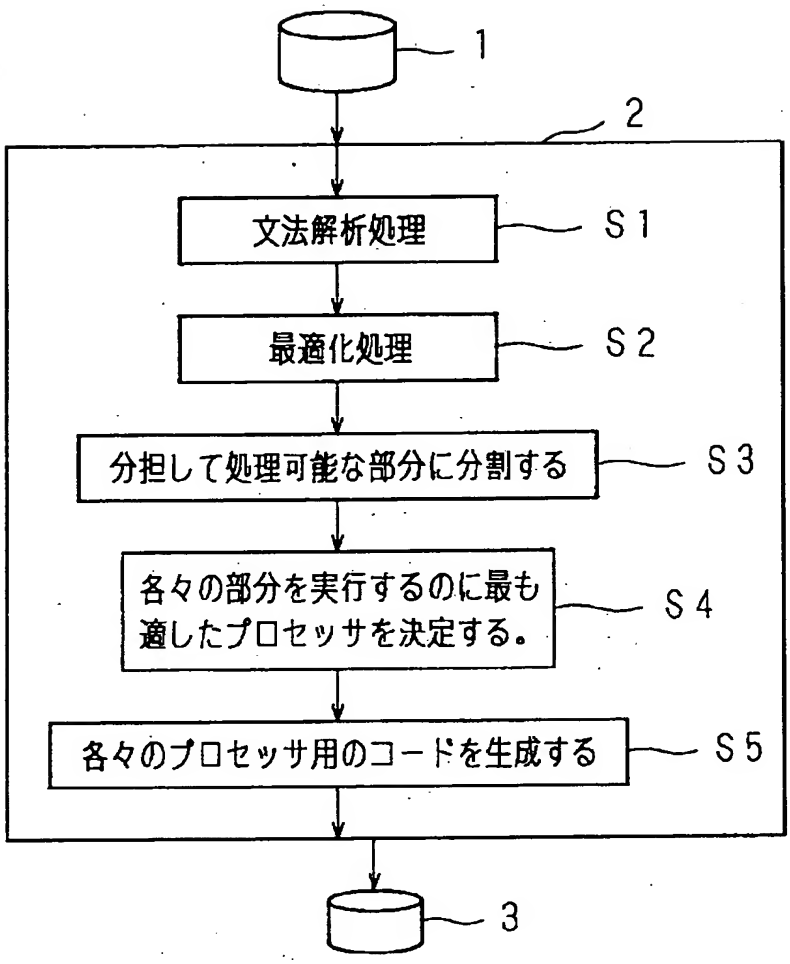
【図５】

```
/*   file name :n4_label$   */
$define VV_INT_ADD 0x10
$define N4_BACK_PROP 0x127
$define N4_USR_FUNC 0x345
        :
```

【図７】



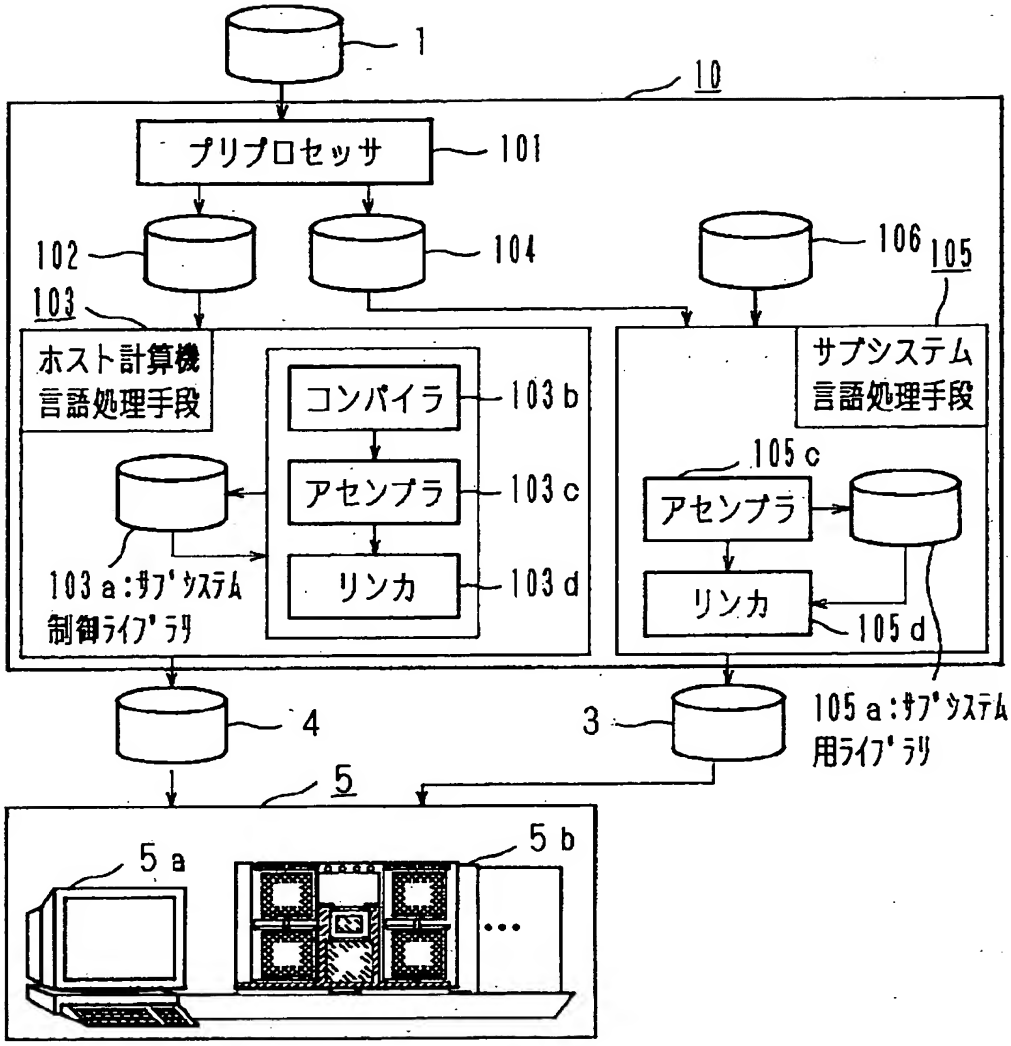
【図２】



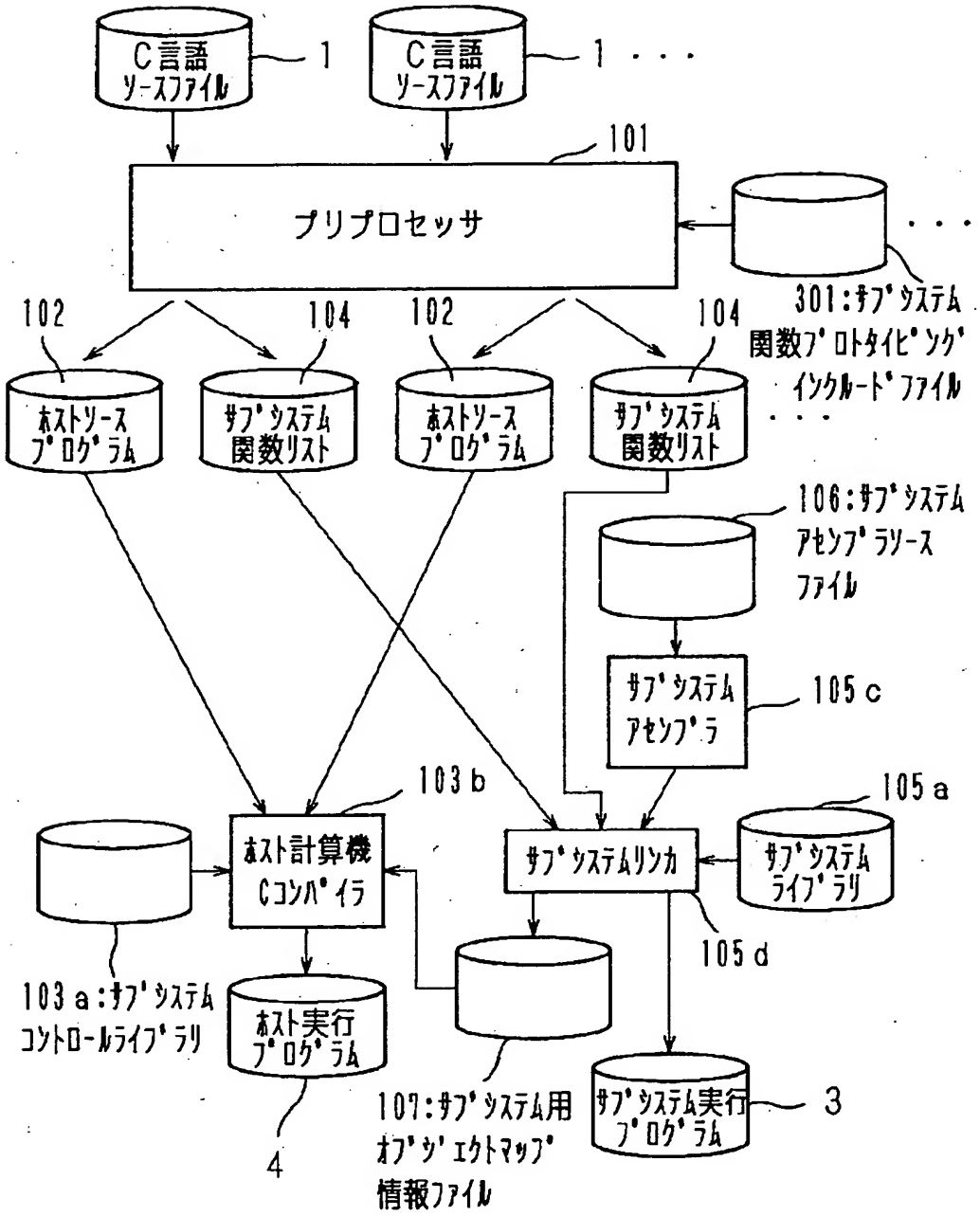
【図９】

```
1: LD      F0,a          ;load scalar a
2: LV      V1,Rx         ;load vector X
3: MULTSV  V2,F0,V1      ;vector-scalar multiply
4: LV      V3,Ry         ;load vector Y
5: ADDV    V4,V2,V3      ;add
6: SV      Ry,V4         ;store the result
```

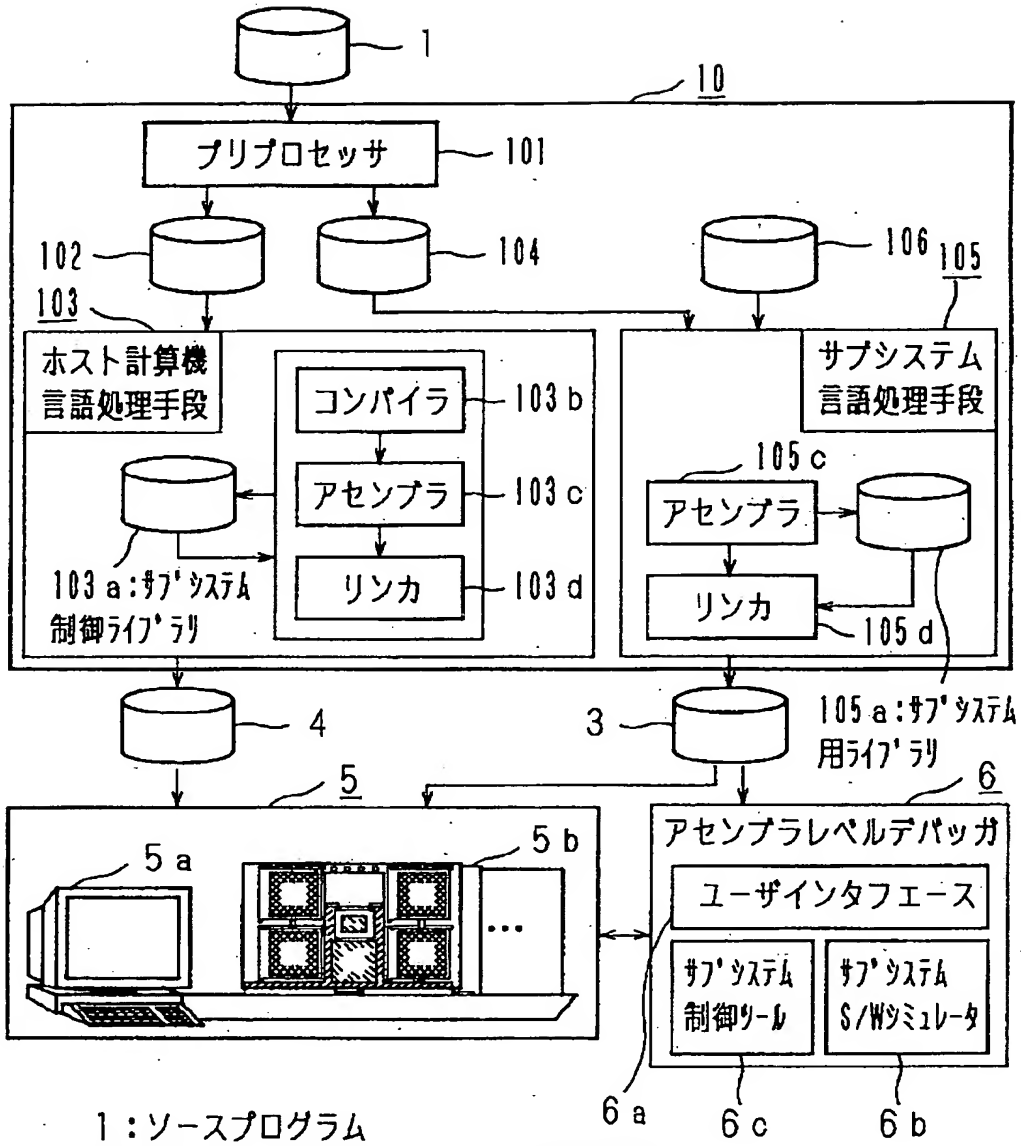
【図３】


1
10
プリプロセッサ
101
102
104
106
105
103
ホスト計算機
言語処理手段
コンパイラ
103 b
アセンブラ
103 c
リンカ
103 d
103 a：ｻﾌﾞｼｽﾃﾑ
制御ﾗｲﾌﾞﾗﾘ
サブシステム
言語処理手段
105 c
アセンブラ
リンカ
105 d
4
3
105 a：ｻﾌﾞｼｽﾃﾑ
用ﾗｲﾌﾞﾗﾘ
5
5 a
5 b

【図４】

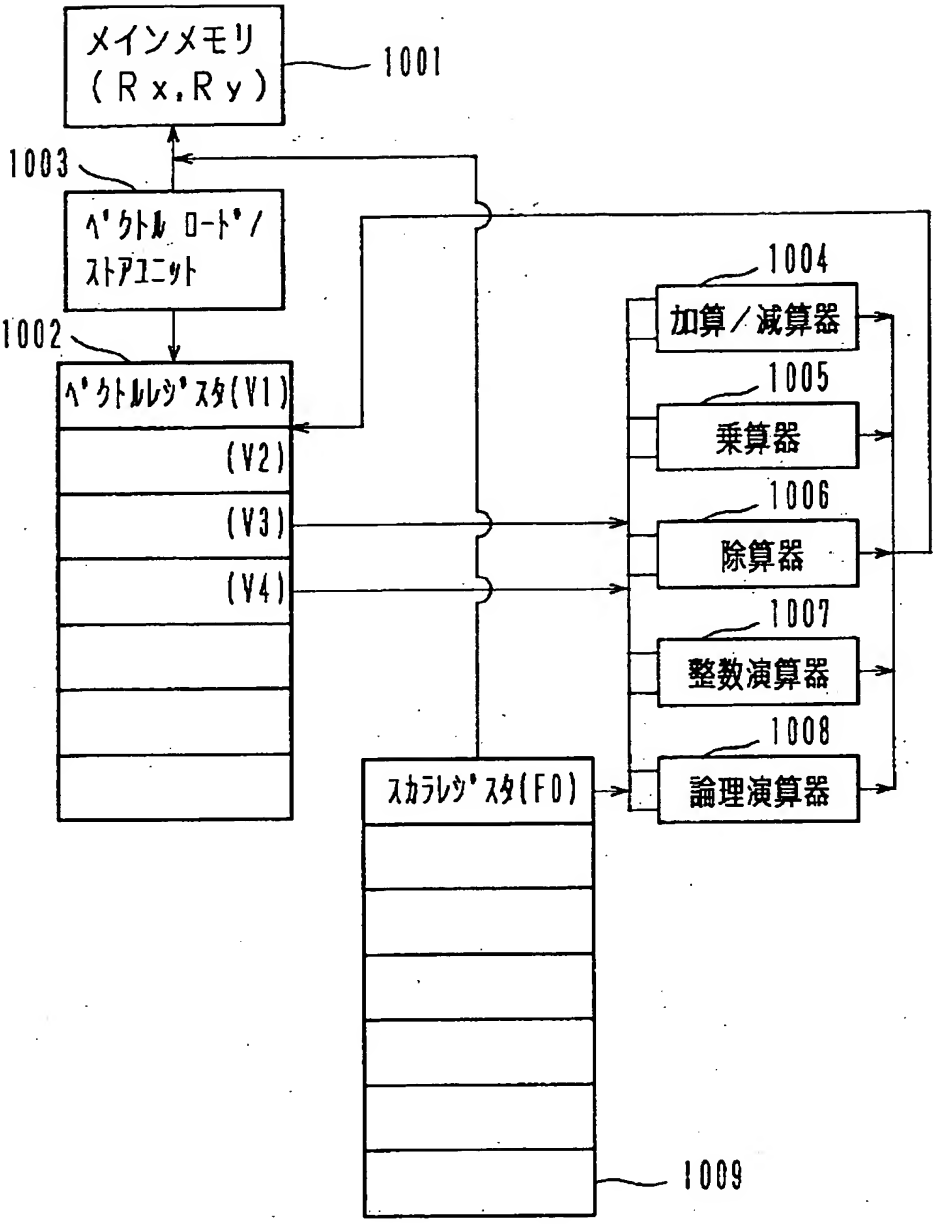

C言語
ソースファイル
1
C言語
ソースファイル
1 ・・・
101
プリプロセッサ
・・・
301:サブシステム
関数プロトタイピング
インクルードファイル
・・・
102
104
102
104
ホストソース
プログラム
サブシステム
関数リスト
ホストソース
プログラム
サブシステム
関数リスト
106:サブシステム
アセンブラソース
ファイル
サブシステム
アセンブラ
105c
103b
105a
ホスト計算機
Cコンパイラ
サブシステムリンカ
サブシステム
ライブラリ
105d
103a:サブシステム
コントロールライブラリ
ホスト実行
プログラム
107:サブシステム用
オブジェクトマップ
情報ファイル
サブシステム実行
プログラム
3
4

【図６】



1：ソースプログラム
5：ターゲットシステム
6：デバッガ
10：言語処理系
101：プリプロセッサ

【図８】



フロントページの続き

(72)発明者　小守　伸史
尼崎市塚口本町八丁目１番１号　三菱電機
株式会社半導体基礎研究所内